Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Real-Time Physics-Based Goalkeeper Animations

Cormac Doyle

April 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer Engineering)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: Cormac Doyle Date: 28/02/2022

Abstract

Cormac Doyle, Master of Science in Computer Science

University of Dublin, Trinity College, 2022

Supervisor: Carol O'Sullivan

By nature, humanoid animation is complex, and the task of creating an interactive humanoid is very challenging when trying to take human motor skills into account. The animation of interactive virtual characters playing sports is a hot topic of research. It has many applications in video games, sports science, and robotics. In this project, a dynamic system that reconstructs motion capture data is demonstrated. This system is applied to a virtual goalkeeper, and uses limited motion capture data combined with the target location of a soccer ball to achieve physically plausible goalkeeper animations. This system bypasses complex physical simulations, and takes advantage of the short burst of dynamic motion required in interactive environments. Along with this system, pose-based animation is utilised to enable the goalkeeper to catch the ball using inverse kinematics. Using the combination of the reconstruction of motion capture data and pose-based animations, a real-time physically plausible system for goalkeeper animations is presented.

Contents

1 Background/Introduction **5**

2 Literature Review **7**

2.1 Physics-based Simulation 7

2.1.1 Physics Principles 7

2.2 Data-Driven 8

2.3 Dynamic Driven 9

2.4 Dynamic and Data-Driven Hybrid 9

2.5 Goalkeeper Motions 10

3 Relevant Concepts **11**

3.1 Character Rigging 11

3.2 Motion Capture Data 11

3.3 Physics Engine 13

3.3.1 Rigid Body Dynamics 13

3.3.2 Rigid Body Collisions 13

3.3.3 Constraints 14

3.3.4 Iteration Solvers 15

3.3.5 Configurable Joints 15

3.4 Inverse Kinematics 15

3.5 Pose Based Animation 16

4 Environment Setup **18**

4.1 Physics Engine 18

4.2 Character and Motion Capture Data 18

4.3 Soccer Goal 19

4.4 Ball Shooting Mechanism 19

4.5 Animation Controller 19

5 Methods **21**

5.1 Inverse Kinematics . . . 21
5.2 Active Ragdoll . . . 21
5.2.1 Physics . . . 21
5.2.2 Rigid Bodies . . . 22
5.2.3 Degrees Of Freedom . . . 22
5.2.4 Target Rotations . . . 22
5.2.5 Balancing System . . . 22
5.3 Transition Motion Synthesis . . . 23
5.3.1 Reconstructing Existing Motion Clips . . . 24
5.3.2 Maximum and Minimum Bounds . . . 25

6 Implementation **28**
6.1 Collision Mesh . . . 28
6.2 Reference Animations . . . 28
6.3 Reaching For the Ball . . . 29
6.3.1 Inverse Kinematics . . . 29
6.3.2 Interpolation . . . 31
6.4 Ball Catching Mechanism . . . 31
6.5 Active ragdoll . . . 32
6.6 Transition Motion Synthesis . . . 33
6.6.1 Reconstruction of Animation Curves . . . 33
6.7 Intelligence Level . . . 36
6.7.1 Testing Framework . . . 36
6.7.2 Intelligence Level Adaption . . . 36

7 Results and Evaluation **38**
7.1 Catching Mechanism . . . 38
7.2 Plausibility . . . 38
7.2.1 Active Ragdoll . . . 38
7.2.2 Transition Motion Synthesis . . . 39
7.3 Intelligence Levels . . . 40

8 Conclusion **41**
8.1 Contribution . . . 41
8.2 Future Work . . . 42

1 Background/Introduction

1. **Motivation**

   Traditionally, an animation is achieved by creating each frame in a sequence, resulting in smooth animation. This results in a method that requires a skilled artist, is time-consuming, and can potentially result in non-plausible results. Keyframe animation is also only suited to the environment it is designed in and cannot interact dynamically with changes to the environment.

   Due to the complex nature of human physiology, using physics to achieve plausibly animated and interactive humanoid characters has proven difficult, but comes with many benefits. Using a physics-based character system, humanoid motion can be achieved that will react uniquely and dynamically in different situations. The quality and capacity of interactions depend on the system and methods used.

   The realistic motion of interactive characters is highly desirable in modern video games, simulations, and virtual reality. Virtual characters have an infinite amount of motion sequences, interactive capabilities, and visually natural movements. This will keep the players more engaged and interested and result in a more plausible, intriguing final product. It is also of interest in real-life applications such as robotics, which rely on physics principles.

2. **Objective**

   The research of sporting motions and computer graphics is a hot topic in today's age. Recently deep learning has revolutionized interactive animated characters. While deep learning techniques are very complex and require lots of data, setting up an interactive virtual character is a captivating problem.

   A situation where a character interacts with its environment is a goalkeeper in a soccer game. With the objective of blocking the ball going into a goal, goalkeepers perform dives and jump to either catch or block the ball. To use key-frame animation to animate a goalkeeper saving every type of possible shot at goal would be very time-consuming and expensive. In this project, I will a variety of approaches to create

a virtual character using physics-based techniques to perform plausible goalkeeper responsibilities.

(a) **Environment Setup**

First, an environment must be set up that gives the goalkeeper an opportunity to save the ball. This would require shooting the ball accurately at the goal, hitting the goal at a user-specified position.

(b) **Save Shots Of Varying Positions**

Setting up a virtual character so that it can block the ball or catch the ball in a plausible fashion, from any position in the goal. This can be measured by testing edge points and unusual spots in the goal.

(c) **Plausible Animations**

The goalkeeper should save the ball in a plausible, realistic manner. This could be a simple save, such as the goalkeeper reaching their hand out and stopping the ball, or more complex maneuvers like diving across the goal with their arms out to stop the ball. If the goalkeeper slides across the ground or limbs become detached from the body this goal will not have been achieved. I am aiming for smooth realistic animation that mimics real-life reactions.

3. **Summary**

Chapter 1 will be a review of the current research of physics-based animation. I will relate each method to this project and access its feasibility in implementing it for a virtual goalkeeper. Chapter 2 will review some relevant concepts. These are topics that are necessary to understand the implementations and techniques used throughout the project. Moving on, I will give an overview of the Environment Setup. These outline the rules of the game that will remain constant when reviewing different physics-based animation techniques. Following, the next two chapters will cover the methods used and implementation of them. Finally, the results will be evaluated and discussed, followed by a conclusion.

2 Literature Review

Physics-based characters have achieved realistic movements decades ago, such as leaping and tumbling seen in (1). The task of animating and simulating real-life characters has been approached in a variety of different ways (2), (3). The following is a review of the most relevant methods to the objective of this project.

2.1 Physics-based Simulation

2.1.1 Physics Principles

One of physics-based animation's most practical applications in the real world is in robotics. This section will review the utilization of physics-based animation when applied to robotics.

In (4), a physics-based system is defined as a system that relies on the laws of physics to make predictions. Mass and momentum are highlighted as the most important physics principles for robotics, along with more straightforward principles like object collisions and object rotational degrees of freedom. These are the principles that will be focused on and are most relevant to this project.

The objective is a human motion controller for virtual goalkeeper animations. When designing a human motion controller, there is a trade-off between functionality and visually natural animation. These goals can be conflicting. Physical simulations with lots of hyperparameters can require endless tweaking, and while can achieve functional results, produce motions that do not replicate human movement. Using muscles and tendons (5) to drive a humanoid character has been proposed to produce more natural motion instead of directly controlling torque applied to joints like seen in (6). Muscle-actuated simulations are very complex and computationally expensive and are not within the scope of this project. For a virtual goalkeeper that achieves visually plausible results, natural motion will be prioritised, while making dynamic decisions based on physics principles. Physically valid, or an accurate robotics simulation is not in line with the objectives of this project.

Using real-life motion capture data is also utilised in robotics (4). This makes logical sense

when trying to recreate natural motion. When designing a motion controller, motion data can be used in many ways, depending on the objective of the controller. Controllers that rely heavily on motion data achieve very visually natural results but are limited to situations which the motion data is relevant, and cannot handle unexpected situations. On the other hand, controllers which rely less on motion capture data, and are based solely on dynamic solutions may handle new situations, but are very difficult to set up and produce less plausible results. The generalization of human motion is not within the scope of this project, with the human motion limited to goalkeeper animations. Therefore, a motion data-dependent approach is viable, depending on the motion capture data available.

This leads us to the next three sections, where human motion controllers which use data-driven, dynamic-driven, and a combination of both will be reviewed.

2.2 Data-Driven

When creating virtual characters, data-driven animation requires a large collection of motion capture data for the motion controller to rely on. Through the use of a combination of networks to reinterpret data and interpolate the movements on a frame-by-frame basis, plausible motion can be achieved. An interactive motion controller can then be designed based on user input or changes to the environment.

The advantage of data-driven methods is that motion-capture animation is already physically plausible because it is taken from real-life motions. Therefore, if motion capture data can be applied to a virtual character it will look natural, and physical simulations can be bypassed. The disadvantage of data-driven animation is that it is expensive and costly to collect sufficient amounts of data. Motion controllers using only data-driven methods are also limited by the motion capture data available.

In (7) computer vision is used to track human body parts using a 2D model to project into 3D human postures. Demonstrating a feasible way to capture motion cap data into an animated humanoid, from a single calibrated camera. This results in a visually natural humanoid character, that's motion is based on real-life motion sequences. This method would work for goalkeeper animations and overcome the price of most motion capture methods, however, it would still require a lot of motion capture data of a goalkeeper saving the ball in all positions of the goal, which would be very time consuming to collect.

A more recent approach has been the use of recurrent neural nets. In (8) a real-time character control system approach is taken using a Phase-Functioned Neural Net which has been trained for human movements like walking, running, and jumping over obstacles. This allows the character to adapt to a variety of terrains and jump or leap over obstacles of varying heights. Neural Net approaches are very powerful but are very difficult to train and

require a huge amount of data.

Readily available goalkeeper motion capture data is limited, therefore a data-driven approach would not yield the same results using these methods due to lack of data.

2.3 Dynamic Driven

Dynamic Driven data uses forces, torques and, momentum to achieve the movement of virtual objects. Due to the complexity of humanoid characters, methods that rely only on dynamic-driven techniques are very difficult to achieve plausible results. Therefore a reference motion or sequence of poses is often used to set targets for the dynamic controller.

In (9) motion cap data is used for flat terrain, but when unexpected terrain is introduced a dynamic driven approach is used. Using a stream of poses, an automated response to changes in the environment can be generated. While not physically valid, realistic, stylized motions are achieved. This technique can also be used for other locomotive actions like running, jumping, and cartwheeling in a cartoony fashion. This approach does not force physical consistency, but physically plausible reactions to changes in the environment.

Similarly, using a rag doll that mimics the poses of motion capture data, combined with dynamic simulation, an interactive character is set up. In (10) this technique yields impressive results with multiple characters interacting in various scenarios. This method is very relevant to this project which tackles the problem of achieving physically plausible responses to a goalkeeper's responsibilities.

2.4 Dynamic and Data-Driven Hybrid

In (11) motion capture data is used to achieve animated characters using a collision model which can also interact with their environment and other characters. Similarly in (12) algorithms for adjusting momentum and force are applied to the dynamic motions of humanoid characters. It achieves plausible results using physics controlled through user input and motion capture.

Functional approaches like those seen in (13) and (14) which use rigid bodies that follow a reference motion will be reviewed. This approach abides by physical principles, however, requires complex balance systems. While this method produces an accurate physical simulation, it can also result in plausible natural animation. This approach will be reviewed for the efficacy of visually natural results, while also providing a physical simulation that provides functional human motion.

In (15) visually natural motion is achieved using a dynamic system that generates physics-based responses. This system uses transition motion synthesis which computes a blend between motion capture data clips. Dynamic systems are often only needed for a short burst in response to some change in the environment, and this system takes advantage of that. This approach, similar to this project is not a physical simulation but achieves visually natural results using reference motion capture clips combined with a physics-based dynamic response. A transition motion synthesis approach is highly relevant to goalkeeper animation, since, a goalkeeper's reaction to a shot will require a short burst of dynamic response.

In (16) uses a data-driven method and physics-based motion synthesis to achieve the task of dribbling a soccer ball forward, dribbling side to side, and shooting, all at the control of the user. Using a synthesized method, the player can carry out these actions in various scenarios. This project's method does not however have a goalkeeper, which is the problem I am taking on in this project.

2.5 Goalkeeper Motions

Goalkeeper motion is complex with a variety of different ways to save a shot depending on the target location of the ball. If the ball is at the edge of the goal, a long dive combined with a catch is necessary, whereas, if the ball is shot directly at the goalkeeper, only a short step and catch is necessary.

In (17) the biomechanical response of 11 goalkeepers diving motions to shots at 3 different heights, located a short or long distance from the goalkeeper was studied. The center of gravity, joint angle, ground reaction force, and joint torque of the diving motion was obtained. During the take-off part of the diving motion of a goalkeeper, it was found that the counter-side leg to the ball controls the magnitude of power, and the ball side leg controls both the magnitude and direction of power in order for the goalkeeper to dive directly towards the ball. This study also provides the change in joint angles for the hip, knee, and ankle of the counter side and ball side legs, when a ball is in the near upper, near middle, and near lower sections of the goal.

These findings are crucial and the results of this study can be used when building the dynamic motion controller for goalkeeper animations.

3 Relevant Concepts

3.1 Character Rigging

Character Rigging is the process of applying animation to a character model. The task of moving complex characters in a plausible manner is difficult and a demanding area in computer animation. Animating models contained as a polygon mesh is impractical, therefore the character is broken down into a skeleton of hierarchical bones, by which the movement of the surrounding mesh can be deduced. This hierarchical structure is rooted at the hips with each bone attached to it being sorted into a hierarchy and inheriting its parent's transforms and rotations. The bone structure of a virtual character can be seen in 3.1

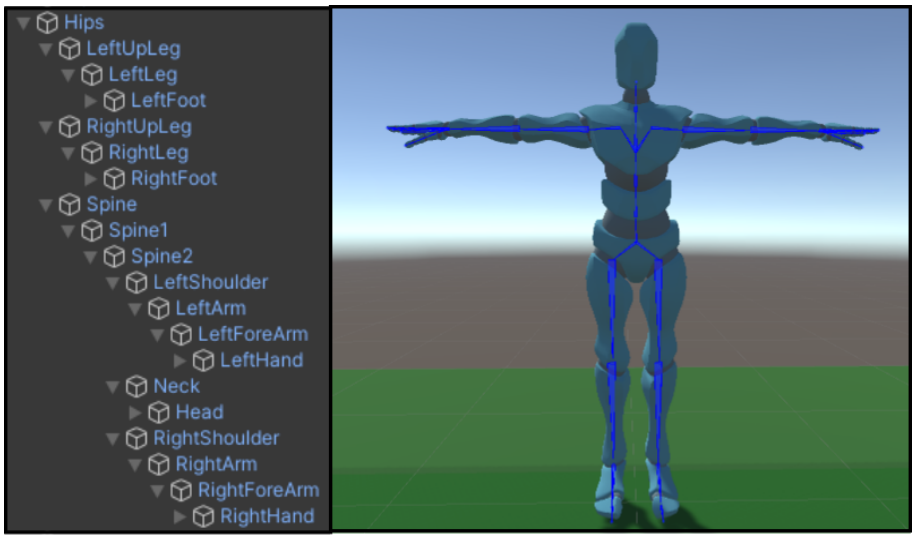

Figure 3.1: Bone Structure Hierarchy

3.2 Motion Capture Data

Motion Capture Data is the result of recording movements of objects or people. When using data-driven animation methods, motion capture data is essential.

Motion Capture Data is stored in a variety of ways. A common way of storing motion

capture data is using animation curves.

An animation curve holds multiple keys which are control points the curve passes through. The construction of the animation curve is dependent on the tangents of each key. Tangent options include smooth, linear, constant, and free connections, and can be set for the left or right side of each control point. 3.2 shows the difference between animation curves with key-frames set to smooth versus constant. Animation curves are also used to create non-linear interpolation paths.

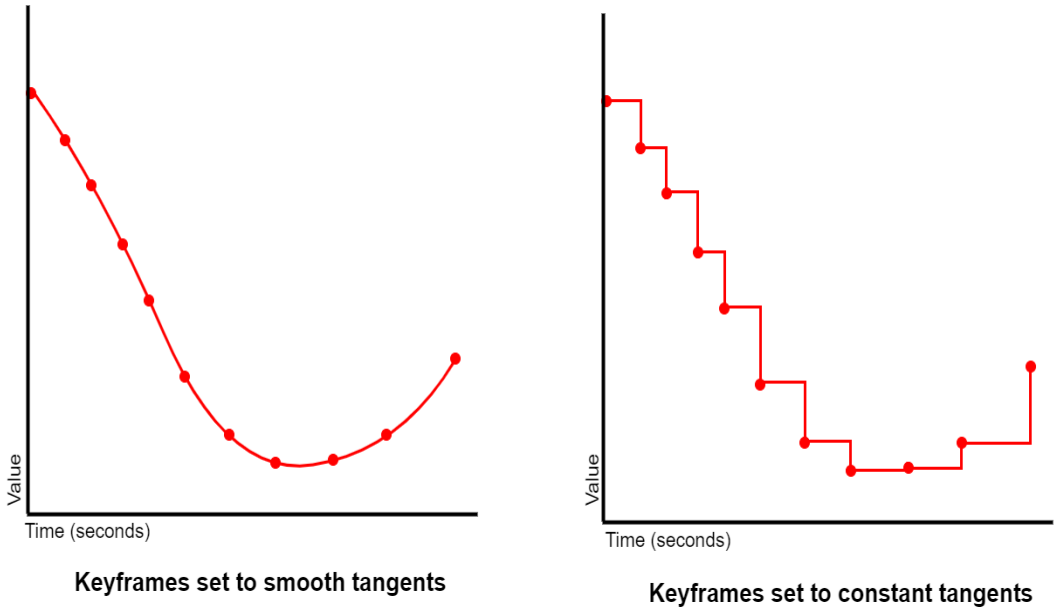

Figure 3.2: Caption

For humanoid models, an animation curve is stored for every bone in the human body structure. Each animation curve represents the bone's position and rotation for each frame for the animation duration. Motion capture data also contains animation curves for the root position and root rotation of the character, which could be considered to be applied to the hips, which is the root of the hierarchy.

Animation curves holding positional information have three curves, an x , y, and z position curve to produce a transform vector for each frame.

Animation curves for rotations can be stored in two ways, Euler Angles or Quaternions. Euler Angles represent rotations using three fields, yaw (z), pitch (y), and roll (x). The advantage of using Euler angles is they are easy to interpret but are susceptible to problems like Gimbals Lock, which results in a loss in degree of rotation when two axes are aligned. Quaternions consist of four fields, x, y, z , and w. Quaternions use a complex numbers coordinate system and therefore these fields do not represent angles or axes directly, making them more difficult to work with. Quaternions can describe any 3D rotation, are computationally efficient and do not suffer from Gimbals lock.

When using animation curves for rotational interpolation, Quaternions will always produce the shortest path between two rotations, however, because of this, cannot interpolate between rotations larger than 180 degrees. They are also difficult to manually manipulate due to their complex coordinate system. Euler angle interpolation is more common and easier to work with.

3.3 Physics Engine

When implementing physics-based animation often times a pre-built physics engine is used to simulate physical bodies. Using a physics engine, rigid body physics, collisions, articulations, joints, and character controllers can be easily applied and configured into a virtual environment. In most applications, rigid bodies are most commonly used and are the most commonly simulated object in modern video games when trying to achieve a realistic physics simulation. A brief overview of how most game engines implement the physics of rigid bodies is provided.

3.3.1 Rigid Body Dynamics

Rigid Body dynamics mimic the real-life motion of solid, inflexible objects. The motion of a rigid body can be described by Newtonian mechanics which were founded upon by Newton's Three Laws of Motion:

1. Law of Inertia
2. Law of mass, and acceleration: F = ma
3. Law of motion: for every action, there is an equal and opposite reaction

Once an object is setup, forces are applied based on these laws, the new position and rotations are updated, the output is displayed, and the process is repeated. This results in a dynamic experience that simulates real physics.

3.3.2 Rigid Body Collisions

In order to realistically simulate physics, physics engines must detect collisions. This adds another step to the process described above whereby for each moving object a collision must be checked for, and resolved, otherwise rigid bodies would just move through each other. There are two types of collision detection, discrete and continuous.

Discrete Collision Detection

Discrete Collision detection handles collisions between rigid bodies given a static snapshot of a simulation. It uses a selection of algorithms to check for intersections between every object

in the scene to determine collision points. Discrete Collision Detection is very precise for slower-moving objects, however, do not account for tunneling. Tunneling arises when an object is moving so fast it may not intersect with another object, but move through it 3.3.

Continuous Collision Detection

Continuous collision detection overcomes tunneling by taking previous and current frames into account. It assumes linear motion between frames and attempts to compute the time of impact. Once the time is computed, it goes back in time to compute the collision point. Continuous collision detection handles more use cases for collision detection, but is more computationally expensive and should be saved for fast-moving objects.

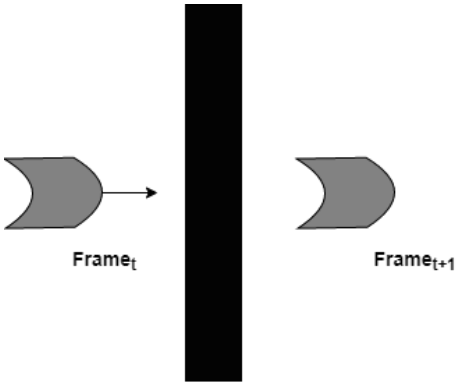

Figure 3.3: Continuous Collision Detection

Using a discrete or continuous collision detection method, collision points between dynamic objects can be found. However, the reaction to a collision still needs to be calculated. This leads to the next rigid body simulation step, applying constraints.

3.3.3 Constraints

Constraints define restrictions on the motion of rigid bodies. Constraints are necessary to account for joints, and non-penetration constraints, such as reactions to collisions. When calculating constraint forces, the constraint force must be calculated before applying the simulated motion. Using Impulse dynamics, a computationally efficient system can be used to calculate the resulting forces in a physics engine. Impulse dynamics can struggle to compute resting forces compared to force-based approaches but its simplicity and efficiency overcome the drawbacks. An impulse is the integral of a force with respect to time:

$$P = \int F dt$$

which is equal to the change in momentum over that time period. When two rigid bodies collide, according to Newton's third law of motion, they apply equal and opposite forces to one another. Using the change in momentum, the resulting forces can be calculated.

Rigid Body Dynamics and collision detection can get computationally expensive when there are lots of rigid bodies in the scene. Because of this, complex objects often use simpler, invisible meshes, for collision detection, which will not cause performance issues.

3.3.4 Iteration Solvers

When dealing with the interaction between a large number of rigid bodies, calculating the exact position, velocity, and force for every rigid body in every frame, is very computationally expensive. Iteration solvers are used to make approximations that get better with every iteration. A low number of iterations will give a less accurate approximation but can increase performance greatly. In physics-based animation, an accurate approximation of physics is necessary for plausible results, but good performance is also necessary for smooth animation. This creates a trade-off between the number of iterations used by the physics engine and smooth animation. With powerful enough computer hardware both can be achieved.

3.3.5 Configurable Joints

Configurable joints are a physics constraint that can be applied to attach rigid bodies at a fixed point. They allow properties of angular joints to be configured. The most important configurable properties related to this project include:

1. Connected Body. The rigid body to attach to.
2. Anchor. The position in space where the centre of the joint is located.
3. Angular limits for X, Y, and Z motion.
4. Angular X, Y, and Z limit spring. The torque to apply to rotate the joint back to its position. This also includes a dampener field which reduces the torque based on the speed of the joint's rotation. This is necessary to avoid oscillations around the target position.
5. Target Rotation. The rotation are stored as a quaternion that the joint moves to using its drive forces. This rotation is stored relative to its connected body.

3.4 Inverse Kinematics

Given a chain of links, inverse kinematics aims to calculate the rotation of each link to position the end of the final link at a target location.

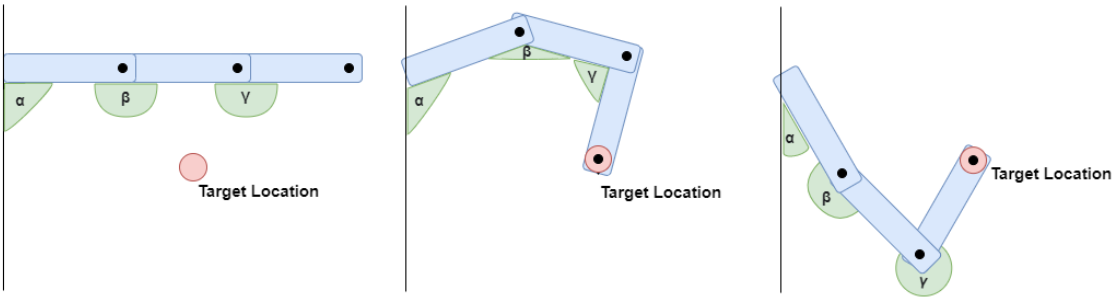

Figure 3.4: Inverse Kinematics

When applying inverse kinematics to a chain of one or two links, a mathematical analytical solution can be used, however when applying inverse kinematics to chains longer than two links, more complex algorithms are necessary. In 3.4 the target rotations of angles in a chain of three links to reach the target location is shown. 3.4 also shows that there is more than one solution to reach the target location. This is important when using inverse kinematics on humanoid models, where the rotations used to achieve the target position matter.

Algorithms for multi-chain links include cyclic coordinate descent, which rotates each link by the difference in rotation to the target and end of chain until the target location has been reached. A Jacobian method can also be utilised. This is a repetitive process, which finds the target rotations by correcting each link angle in increments, using partial derivatives.

3.5 Pose Based Animation

Pose based animation is the interpolation between two character poses to achieve an animation sequence. This is a technique that can be used on top of motion capture data to allow a character to interact with its environment. For example, if a character needs to reach out and grab an object, we only need to adapt the character's arm animation, the rest of the animation will remain the same. In this case, the initial pose will be the arm of the character by its side, and the target pose will be the characters arm extend to grab the object. Instead, of simply setting the target position of the arm to the target rotations, pose-based animation can be used to interpolate between poses to create an animation sequence, as seen in 3.5.

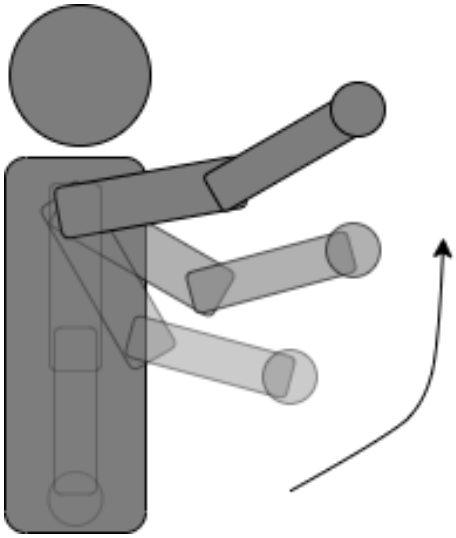

Figure 3.5: Pose Based Animation

4 Environment Setup

4.1 Physics Engine

There is a large selection of physics engines available to use. Unity and Unreal Engine are more suited to video game development, however, provide plenty of out-of-the-box support for physical simulation. While not being an accurate robotic simulation, a close approximation can be achieved. Unity offers a free, high-performance, platform with a physics engine, physics-based rendering, and low-level scripting capability, which suits the needs of this project.

4.2 Character and Motion Capture Data

There are a number of ways to collect motion capture data, however, most of these methods require lots of equipment or use advanced computer vision techniques. The alternative is to use readily available motion capture data from a motion capture database that has been collected by a third party.

Mixamo provides a large database of free, ready-to-use, motion capture data, which includes a variety of goalkeeper animations. These animations include a goalkeeper idle, long dive, high catch, short dive, and low catch. Mixamo's goalkeeper animations will be used as reference animations in this project 4.1.

The character model used will be the default Mixamo character as seen in 4.1. This

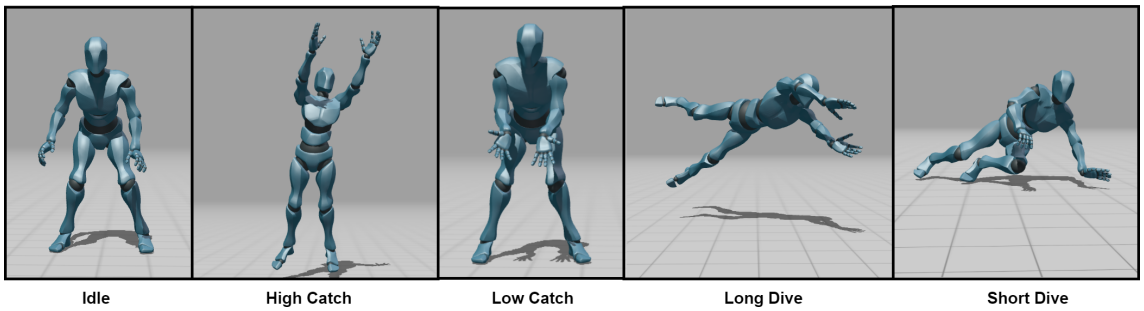

Figure 4.1: Screenshots from reference animations

character model comes pre-rigged and ready to use with Mixamo's motion capture data, which simplifies the character rigging process. Mixamo's default humanoid model has accurate human body dimensions which represent the average human being. This makes it very suitable for a goalkeeper model. Since physics-based animation is being prioritised, the aesthetics of the virtual character will be ignored.

4.3 Soccer Goal

When setting up the environment, real-life soccer goal dimensions were used. The ball is placed on the penalty spot, 11 metres from the goal, to simulate a penalty shot being taken. The soccer goal, pitch, and ball dimensions are shown in 4.2

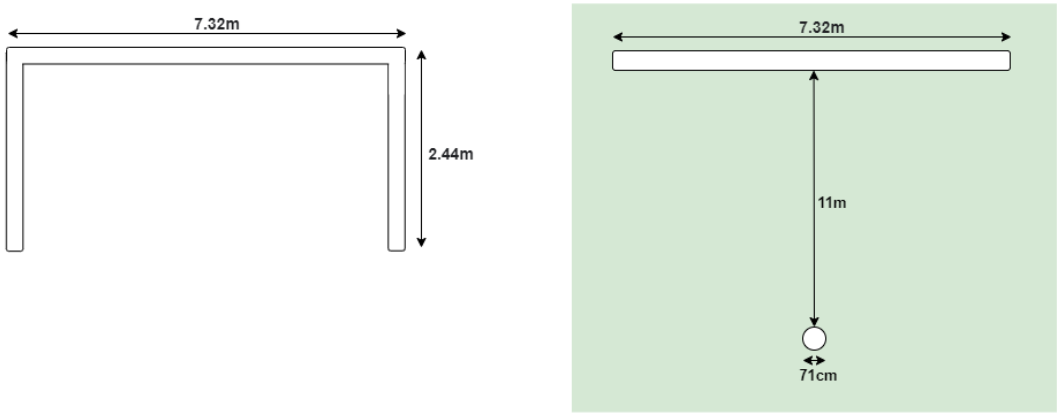

Figure 4.2: Environment Dimensions

4.4 Ball Shooting Mechanism

For testing purposes, a ball firing mechanism, that enables the ball to be shot at any point in the goal must be constructed. To set the target position of the ball a mouse click's location on the screen can be transformed into a world position transform. To achieve this the mouse location and goal's on-screen location is normalised relative to the screen size. Using the mouse click and goals normalised screen position, the world transform of the target position of the ball can be calculated

Once the target position of the ball is calculated a force is applied to fire the ball at the target location.

4.5 Animation Controller

Using Unity's animation controller the appropriate animations can be played starting from the Idle animation. Each animation is mirrored using Unity's animation mirror function for

animations towards the opposite side of the goal.

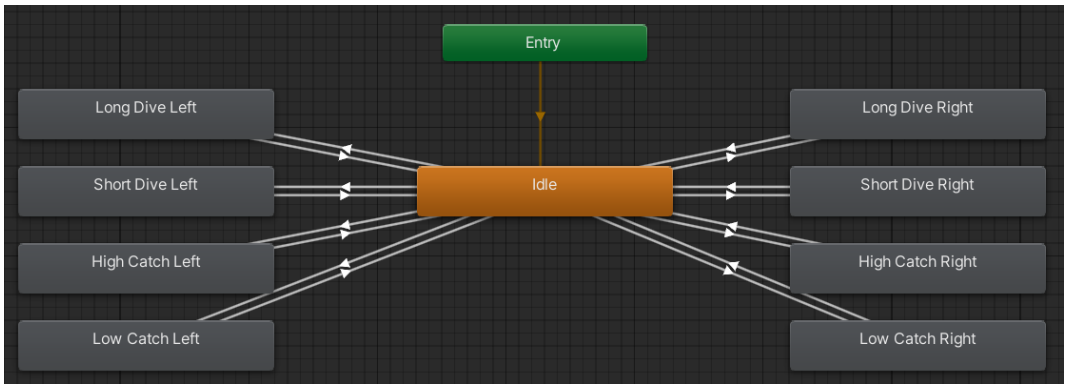

Figure 4.3: Animation Controller

5 Methods

5.1 Inverse Kinematics

Two different methods were attempted when trying to achieve plausible, interactive animations. The first is a more physically accurate approach, the Active Ragdoll approach. The second is less physically accurate and focused more on dynamic methods which prioritises visually natural motion, Transition Motion Synthesis. An overview of the inner workings and theory behind each method will be described.

5.2 Active Ragdoll

The Active Ragdoll approach is a method by which each bone in a humanoid character is affected by the laws of physics. The active ragdoll requires two characters, a physical, active ragdoll character, and a non-physical, standard animated character. The active ragdoll attempts to copy the motion of the animated character. This is done by copying the rotations of each bone in the reference character and applying them to the ragdoll using external forces in real-time. The external forces simulate the forces humans use with their muscles and tendons. Since every bone in the active ragdoll abides by the laws of physics it results in a more interactive character that uses the built-in physics engine to detect and react to collisions.

5.2.1 Physics

As mentioned, every bone in the active ragdoll will be under the influence of physics. It is important that the physics reactions are accurate. The physics engine's iteration solver for force and velocity calculations can be configured to a higher number to increase the accuracy.

5.2.2 Rigid Bodies

A ragdoll is set up by attaching rigid bodies of appropriate mass to every bone on a humanoid character. To constrain the movement of each rigid body a configurable joint with locked motion is attached to its parent in the bone structure hierarchy. For example, we can attach a rigid body to the upper arm and lower arm, however, when under the influence of physics, the lower arm will get detached and roll away from the upper arm unless constrained by an elbow joint. This applies for all bones in the bone hierarchy, rooting from the hips.

5.2.3 Degrees Of Freedom

Next, the appropriate axes of rotation are locked for certain configurable joints. For example, the elbow only rotates around one axis. The degrees of freedom for each axes are also set. This stops limbs from moving to impossible positions, like hyperextension of the elbows. The degrees of freedom of each joint will differ depending on the associated bones.

This setup will achieve a rag-doll humanoid character, where every limb is affected by gravity, but will collapse to the ground in a somewhat plausible manner. The next task is to make the ragdoll active.

5.2.4 Target Rotations

To make the ragdoll active, a reference animation is used which the ragdoll will try to copy. Target rotations are applied to each configurable joint to match the rotation of the equivalent joint in the reference animation. When a target rotation is set for a configurable joint the angular limit spring applies a torque to reach the target rotation. This can lead to oscillations around the target rotation so the joint dampener will have to be configured appropriately.

5.2.5 Balancing System

To make the ragdoll stand, a balance system must be created to keep the character upright. There are many ways of doing this, each with a varying level of complexity.

Freezing the motion of the hips

The simplest way to keep the character upright is to lock the vertical motion of the hips. This will keep the character upright, however, limit its motion significantly. This method has been implemented in some video games, to achieve a cartoony ragdolly motion, but for a physics-based goalkeeper, vertical motion will be essential.

Attaching a rigid body to the hips

By attaching the hips to a kinematic rigid body by a configurable joint a strong limit spring can be set on the configurable joint. This method will apply torque to ragdoll when its hips move away from the target position, but can result in a springy unrealistic effect.

Manual Balance System

In the previous method, the configurable joints spring torque is used to balance the ragdoll. A system to generate the forces required to balance the physical character can also be used. In this system, a force is applied to push the ragdoll to an upright position. A simple implementation of this is very similar to the previous method described and will have the same issues. To overcome the springing effect when applying torque, more complex methods are required.

5.3 Transition Motion Synthesis

Transition motions synthesis computes a blend between motion capture data clips for responses to dynamic situations. This method takes advantage of the fact that for most situations, a dynamic response is not required, and when a dynamic response is required, it is a short burst of response to a change in the environment. Transition motion synthesis also takes advantage of the physically plausible nature of motion capture data. When computing blends between motion clips, two things need to be taken into account: generating a physically plausible motion to fill the gap, and connect the two motion clips as closely as possible.

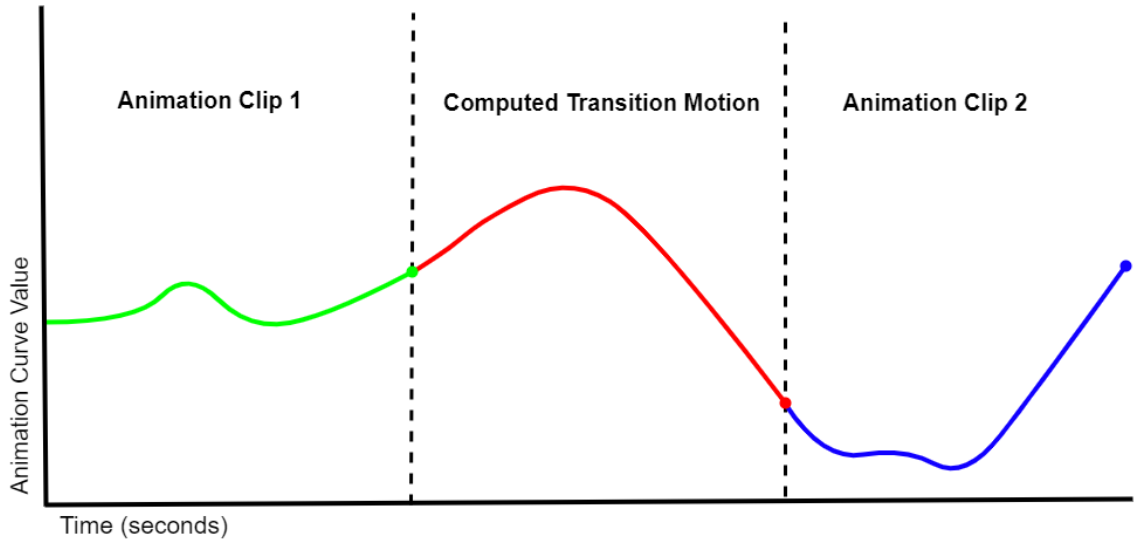

Figure 5.1: Transition Motion Synthesis

In 5.1 an example of a computed transition motion can be seen. When computing a

transition motion curve, many complex methods can be used. The following section will cover the method I will use to reconstruct Goalkeeper animations.

5.3.1 Reconstructing Existing Motion Clips

The same principles used by transition motion synthesis can also be applied to reconstruct a motion clip, by splitting the clip up into sections and reconstructing a section, depending on the situation. In 5.2 the reconstruction of the middle section is shown.

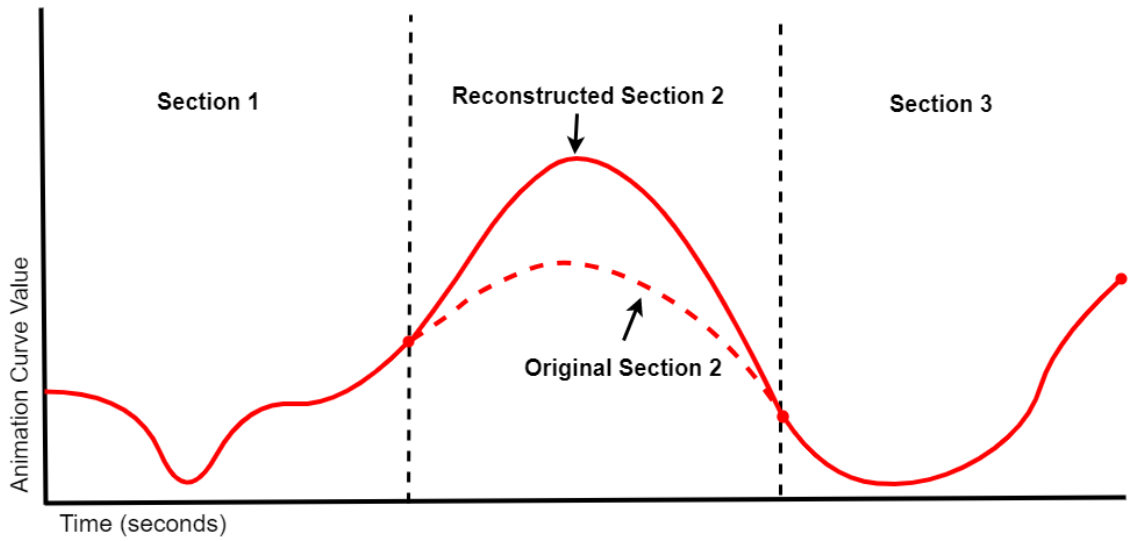

Figure 5.2: Animation Curve Section Reconstruction

The reconstruction of animation curves is done by resetting the values of the keyframes to new target values. In 5.3 the change in key frame values is shown to form the new animation curve. If the tangent options for each key frame is set to smooth, the resulting animation curve will form a continuous line that will pass through all the keyframes. As long as there are keyframes at the beginning and end of the sections, the previous and next sections will connect smoothly. A smooth animation curve connecting the previous and next sections is required to produce visually natural animation.

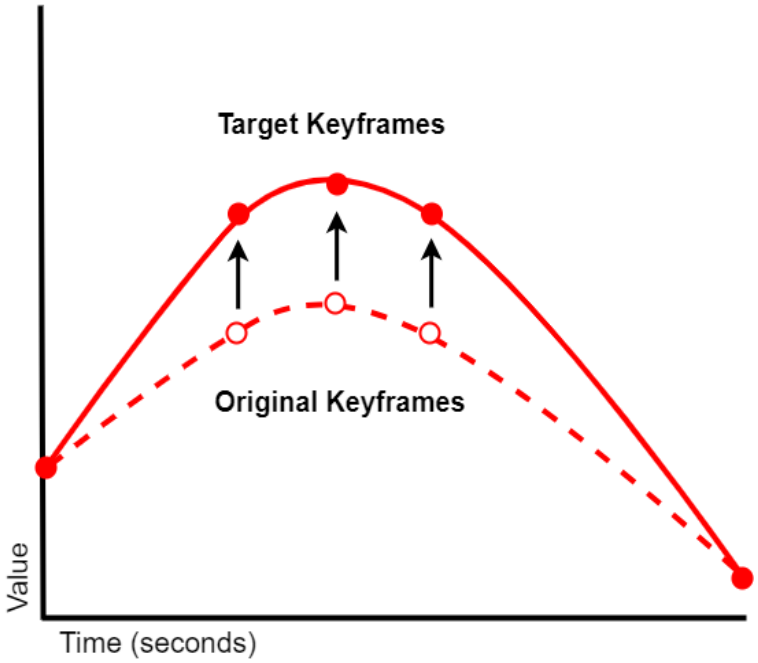

Figure 5.3: Curve reconstruction: Change in keyframes

In 5.3 the y-values of each key frame are increased by a uniform amount. By increasing each key frame by the same amount, the shape of the original curve is kept, but a new peak of the curve can be set. This is important when trying to keep the physically plausible nature of the motion clip. For example, if the shape of the curve shown in 5.3 was representing the motion of a projectile, by the laws of physics its trajectory would follow the path of a parabola. By reconstructing the curve using the method described, the new animation retains the shape of a parabola, while setting a new peak, keeping the motion physically plausible.

5.3.2 Maximum and Minimum Bounds

When reconstructing animations curves, maximum and minimum bounds need to be set, to ensure the shape of the curve is maintained, and the motion remains physically plausible. In 5.4 if the keyframes are moved down too low, the shape of the curve is lost, and the resulting motion may not look natural. Figure 5.3 shows the minimum bound set for the keyframes that keep the shape of the curve.

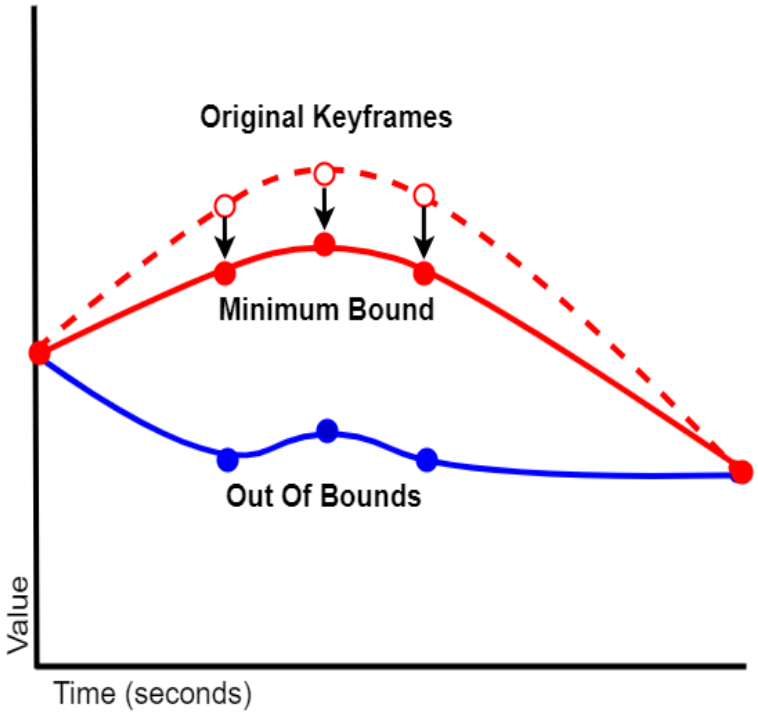

Figure 5.4: Keyframes Out Of Bounds

Figure 5.4 shows how the minimum bound is set. But what about the upper bound? If the change in keyframes is too large, the position or rotation of the game object will move or rotate at unrealistic speeds. In order to check for this, the average velocity between each key frame is checked.

$$velocity = \frac{\Delta x}{\Delta t}$$

If the velocity is beyond physical limits, the change in keyframes is too large and needs to be reduced. The physical limits for average velocity depend on the game object the animation curve is being applied to.

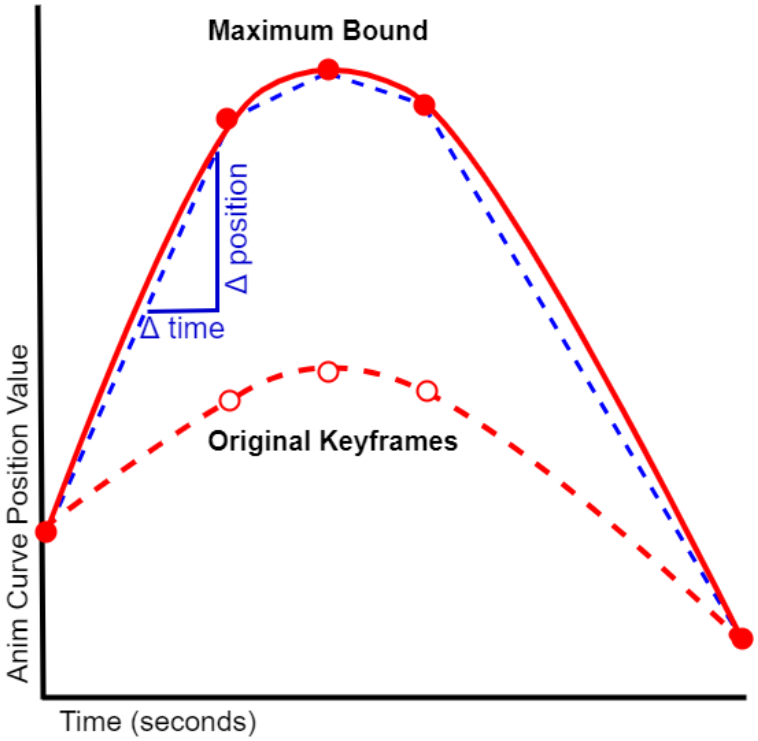

Figure 5.5: Average Velocity between Each Keyframe

5.5 the slope of the blue dashed lines is used to calculate the average velocity between each key frame.

6 Implementation

6.1 Collision Mesh

For the character to block the ball, collision meshes and kinematic rigid bodies must be attached to each body part. Due to reasons discussed in the Relevant Concepts section simplified meshes can be used to improve performance. This way collisions are detected in an efficient way between the character and soccer ball, as opposed to the ball passing through the keeper. The simplified collision meshes are shown in 6.1

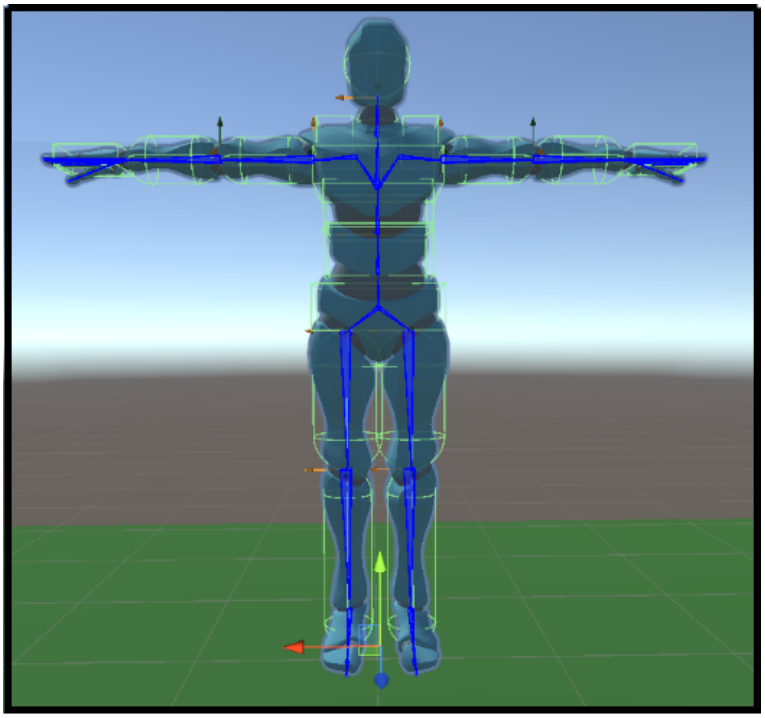

Figure 6.1: Character Colliders

6.2 Reference Animations

When the ball is fired at the goal, the goalkeeper will have to decide what reference animation to play. This decision is made based on the target location of the ball. In this

project, I have not included a reaction time, and when the ball is fired, the goalkeeper knows the target location of the ball. In 6.2 the goal is split up into zones. If the target location of the ball is positioned in the zone, the labelled reference animation will be played. The borders of each zone were calculated by assessing the plausibility of whether the keeper could catch the ball at the edges of the zones for each animation.

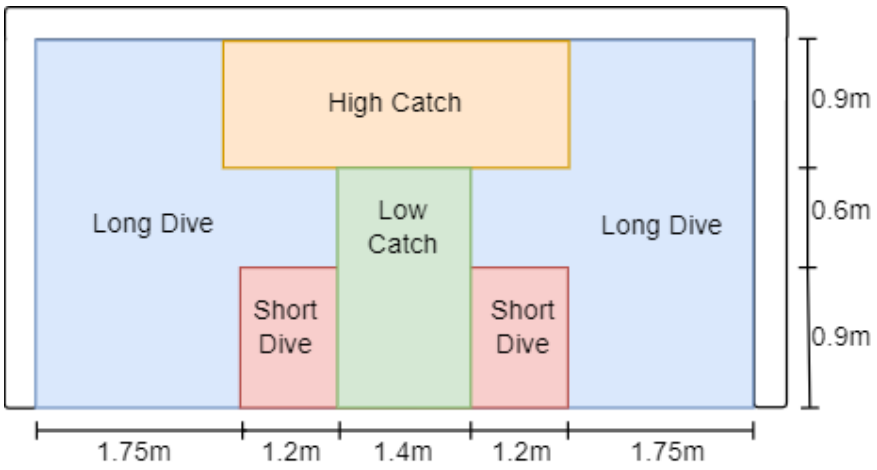

Figure 6.2: Reference Animation for each target location zones

Once each reference animation was assigned a zone, the x and y coordinates of the target location are used to calculate what zone it is going to land in, and the animation controller is used to play the appropriate animation. So far, we have a character that will dive or jump in the approximate direction of the ball but will not save it unless it is fired in the exact position for the animation to save it.

6.3 Reaching For the Ball

To reach for the ball an inverse kinematics system is used to control the arms. Unity's Animation Rigging package can be used to set up a kinematic system for an animated character where the kinematic bodies will override the animation controller. This way the animation remains the same, and only the arms are affected.

6.3.1 Inverse Kinematics

The inverse kinematic system consisted of two parts:

1. **Upper arm and lower arm controller**

   First, a two-bone inverse kinematic constraint is set up with the root link as the upper arm, the middle link as the lower arm, and the hand as the tip of the chain. The target location of the ball is used as the target. This will position the arms with the

hands in the correct location to save the ball, however, because there can be multiple solutions to inverse kinematic targets, the elbows can hyperextend, or collide with the head. To resolve this, a hint transform can be positioned. The hint transform will guide the direction of the inverse kinematic solution toward the desired rotation. An example of the effect of hints on inverse kinematic systems is shown in 6.3. Hint transforms are placed behind and out from each elbow, and the transform is attached to the upper arm so the hint transform will follow the arm throughout the animation.

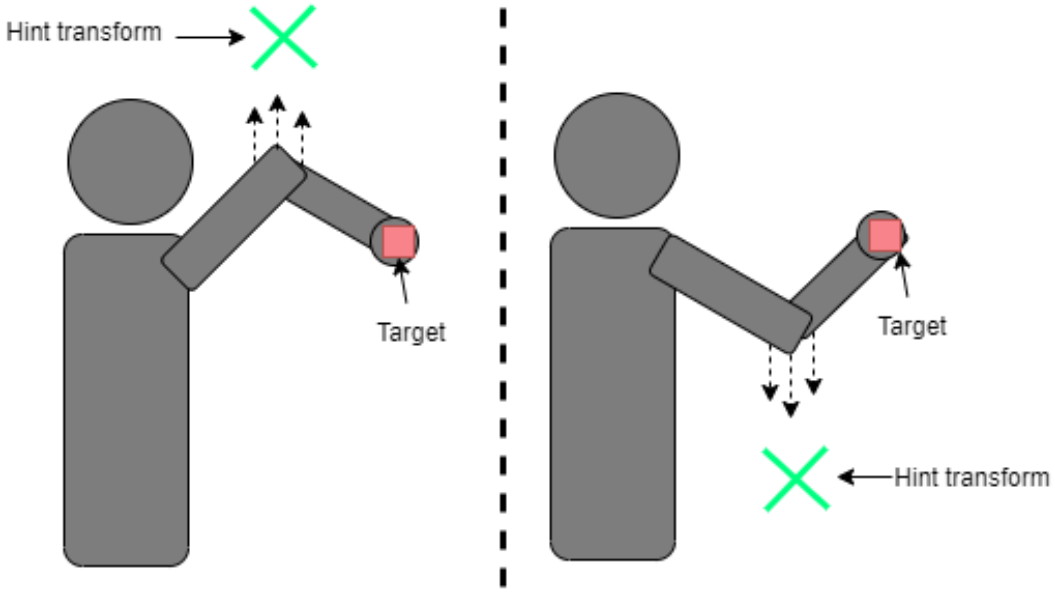

Figure 6.3: Inverse Kinematics Hints

2. **Hands Controller** The upper and lower arm controller will position the hands in the correct position, however, the hands also need to aim at the ball in order to catch it. A multi aim constraint can be used to aim an object at a target. This is used to aim the palms of the hands at the ball. Different target ball positions can be seen in 6.4

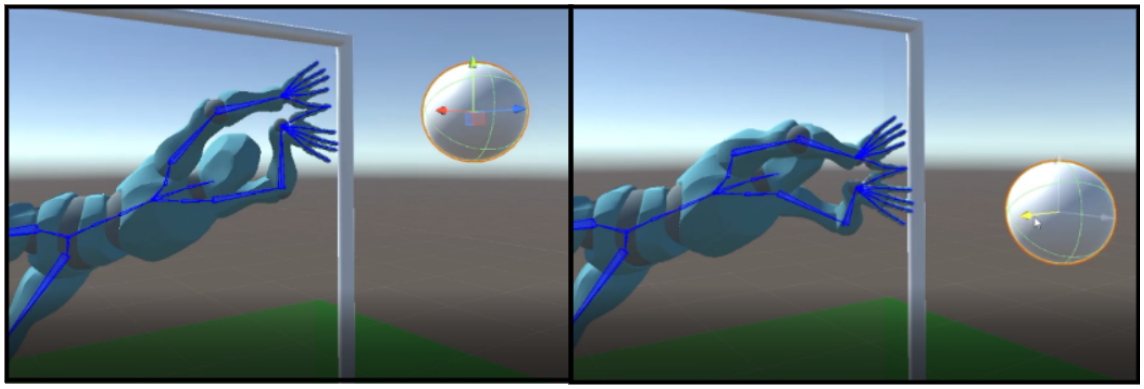

Figure 6.4: Inverse Kinematic System with different target ball positions

6.3.2 Interpolation

The inverse kinematics will position the arms and hands in the correct position, however, the virtual character constantly reaches out for the ball which is not desired. A system to turn the inverse kinematics system to interpolate to and from the target arm positions is required.

Pose-based animation can be used here. We have an initial animation pose, and a target pose to reach. Using pose-based animation we can turn on the inverse kinematics system when the ball is close to the goalkeeper. This will linearly interpolate between the original arm position and the target position, resulting in smooth animation of the goalkeeper adjusting his hand position to catch the ball. Once the ball is caught or moved passed the goalkeeper the inverse kinematic system can be turned off, and the goalkeeper continues with the original reference animation.

6.4 Ball Catching Mechanism

To catch the ball, a deterministic approach was taken. A collision must be detected between the ball and both palms of the hands within a couple of frames for a catch to occur. A simplified collision mesh, seen in 6.5 was attached to the hand to detect collisions with the ball.

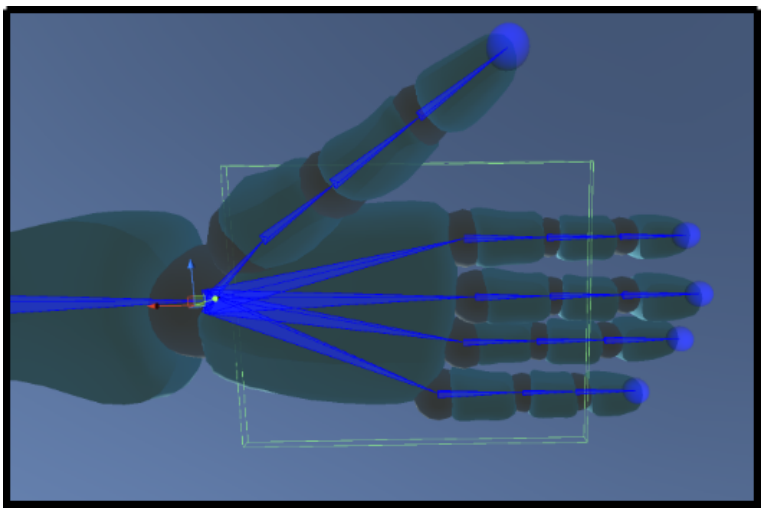

Figure 6.5: Hand Collision Mesh

The soccer ball and hand palm collision mesh both move quickly in the scene, therefore it is their collision detection mode is set to continuous. Once a ball is considered caught, the ball is set to kinematic and attached to the goalkeepers hand.

6.5 Active ragdoll

So far we have an animated character that plays an appropriate reference animation and will catch the ball if it is within reach. Using an active ragdoll, hyperparameters, such as the limit spring on the configurable joints, of the legs could be adjusted to change the trajectory of each diving reference animation to make the ball within reach.

The mass distribution used is seen in table 6.1 and is based on values found in (18).

Body Part	Mass (kg)
Hips	10
Upper Leg	8
Lower leg	5
Foot	2
Lower Spine	10
Mid Spine	10
Upper Spine	10
Shoulder	2
Upper Arm	3
Lower Arm	2
Hand	1
Neck	2
Head	8

Table 6.1: Mass Distribution

To set the target rotations, initially, I tried to take the rotation of each limb in the reference animation and apply it to the active ragdoll, This did not achieve the results I was expecting, and I soon realised the local rotation on the reference animation had to be converted to a configurable joint target rotation. The code to do this had already been written in (19) and achieved the expected results for the active ragdoll to mimic the reference animation.

While the idle animation looked plausible using the active ragdoll, the springy and floppy nature of the active ragdoll caused the ragdoll to not follow the other reference animations properly. The hyper parameters on the configurable joints were altered to try and achieve the most plausible results, however, a predictable, repeatable animation could not be achieved with the active ragdoll.

The initial idea behind the ragdoll was that once it was able to copy the motion of the reference animation, adjustments could be made to the spring torques on the legs, depending on the target location of the ball. Take the long dive animation for example. If the spring torque on the legs is high, in theory, more force would be used to move the legs towards the target location. This extra force would also launch the ragdoll further on takeoff. If the spring torque were low, the goalkeeper would not launch as far, because the

force being applied to his legs was less strong. However, in practice, the ragdoll's animations were unpredictable, and even with a high spring torque applied to the legs, the goalkeeper was not able to launch itself that far.

After tweaking the hyper parameters on the active ragdoll and yielding no plausible results, this leads us to the next method, reconstructing the animation curves, using a transition motion synthesis style method.

6.6 Transition Motion Synthesis

As discussed in the methods section, transition motion synthesis can be used to compute a blend between two motion clips. For this project, a blend between two clips is not necessary, but the reconstruction of a motion clip is. This can be utilised when the ball is out of reach of the goalkeeper.

6.6.1 Reconstruction of Animation Curves

The root position animation curves can be reconstructed to position the goalkeeper in the correct position at the correct time. For example, the reference long dive animation will launch the goalkeeper to the right side of the goal, but if the ball is fired in the top right corner, the ball will be out of reach. The x and y root positions animation curves can be reconstructed to adjust the trajectory of the goalkeeper's dive to make the target location reachable. In 6.6 the appropriate reconstruction can be seen to move the goalkeeper's position to make a target position in the top right of the goal reachable.

In 6.6 the change in x and y keyframes can also be seen to make the readjustment in motion. Ideally, in Unity, the animation curve keyframes within a motion clip would be accessible by script and could be altered dynamically. However, this is not possible in Unity. To overcome this, the following approach is taken:

1. Save the location of all the keyframes in the relevant animation curve.
2. Remove the relevant animation curve from the motion clip.
3. Adjust the keyframes for the reconstructed animation curve.
4. Reconstruct the animation curve using the keyframes through a script at run time.
5. Apply the animation curve to the appropriate game object through a script at run time.

Using this method, the keyframes can dynamically be changed and reconstructed to the desired animation curve. This can be applied to move the goalkeeper to reach any target position in the goal. The position that the goalkeeper reaches does not have to be exactly

correct, because of the inverse kinematics catching system, if the goalkeeper is close enough, the goalie can reach out and catch the ball.

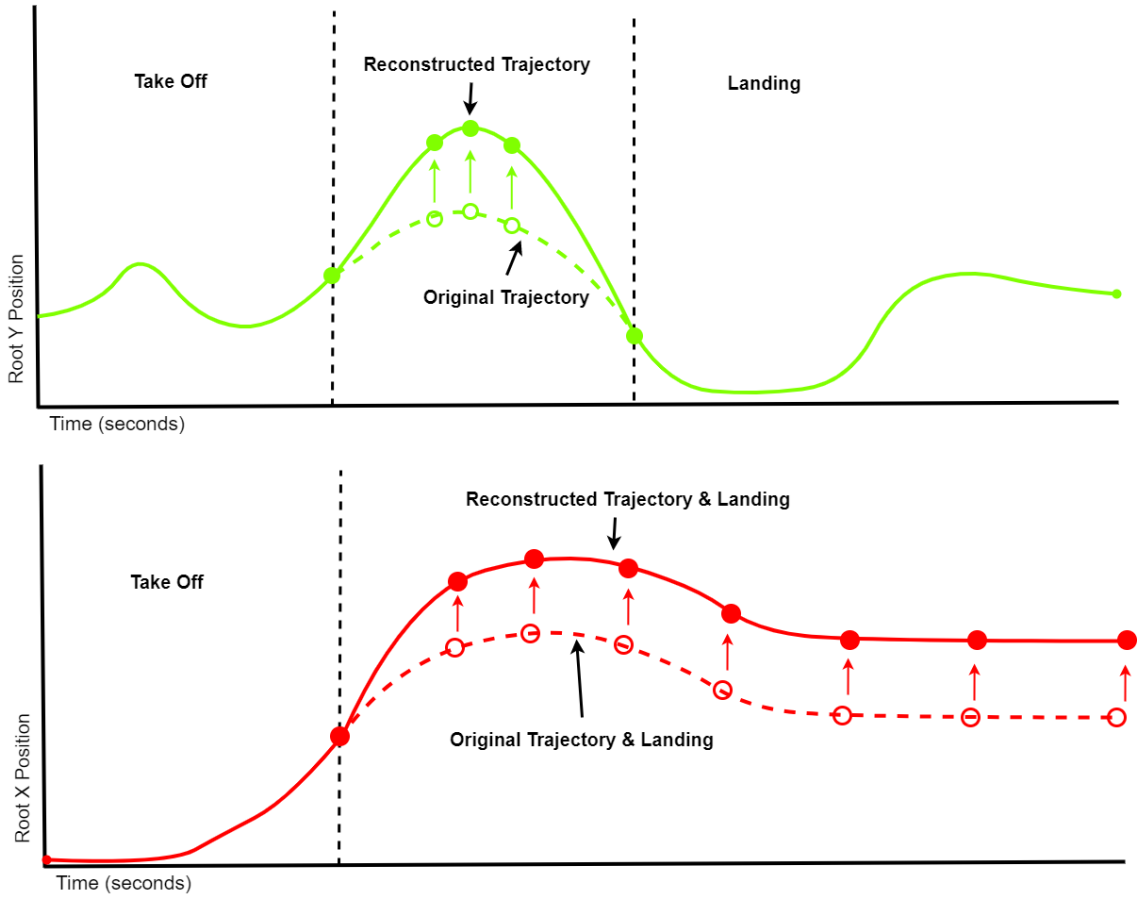

Figure 6.6: Root x and Y position Reconstruction

To set the maximum and minimum bounds, limits were set to ensure the shape of the animation curve is maintained. A maximum limit was also set for the x and y root positions so that the goalkeeper would not dive past the goal, the farthest the goalkeeper was able to jump would be to reach the top corner. This stops the goalkeeper from saving shots that are not fired at the goal, but will still allow him to jump towards the ball, giving the effect that the goalie is attempting to save the ball.

So far, I have shown how I changed the root position so that the goalkeeper is positioned in the correct place to catch the ball. However, this does not look plausible for all cases. We know from (17) where the characteristic 11 goalkeepers diving motion was monitored, that the angle of the body during takeoff changes for different target positions. This paper also shows the different foot placement for the ball side leg for different target positions. Using the characteristics of goalkeeping motion found in this paper, the animation curves for launch angle and foot placement can be adjusted on the virtual goalkeeper.

To adjust the launch angle, a similar approach is taken when changing the root position.

The root z rotation animation curve, which is stored as a an Euler angle, can be easily manipulated using the reconstruction method already described. Based on the height of the ball the launch angle is adjusted appropriately. The launch angle for varying heights can be seen in 6.7.

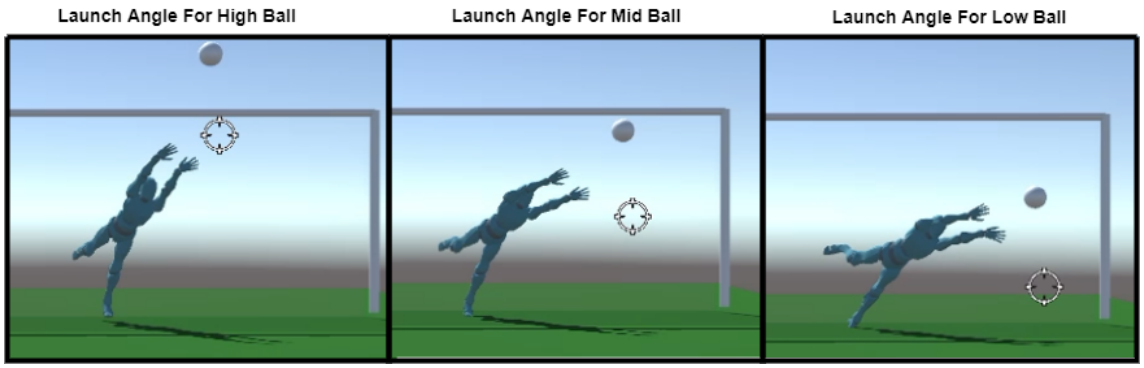

Figure 6.7: Launch Angles

The foot placement of the ball side leg can also be adjusted by reconstruction of the z root rotation for the upper leg transform. From (17) we know that the foot placement of the ball side leg depends on the height of the target position. In 6.8 the change in foot position can be seen based on the target position of the ball.

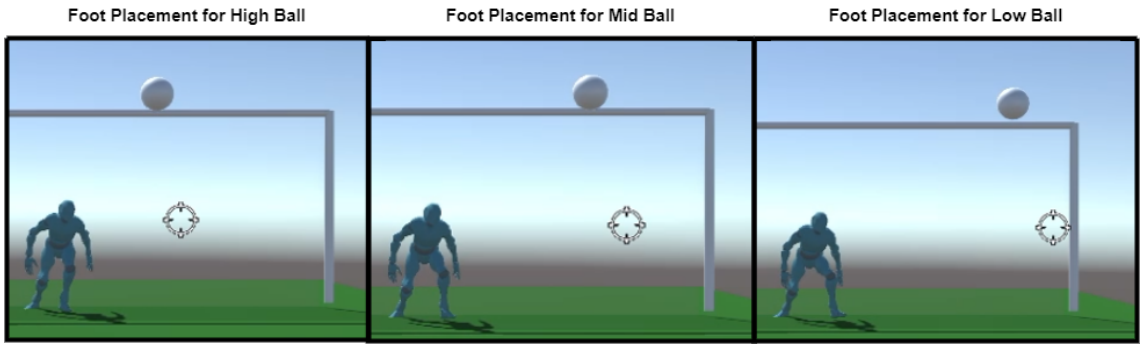

Figure 6.8: Foot Placement Adjustment For Long Dive Animation

Using the same methods, the root position, launch angle and, foot placement animation curves are reconstructed for the high catch and short dive animations. Although there are no characteristics of motion to base the high catch and short dive reconstructions on, the same principles apply. The launch angle is adjusted to aim more towards the target position, and the ball-side leg steps out further, the further away the target position is.

For the low catch animation, there is no jump or die involved. Since this animation only covers a small zone in the goal 6.2, the ball is always in reach, therefore the catching system can reach out and catch the ball.

The goalkeeper's animations are played at a constant speed. This works fine for all the animations except for some situations in the long dive animation. For example, the long dive takeoff speed should be faster if the target position is farther away. Animation speeds can be easily changed using scripts in Unity. Based on the distance to the target position, the takeoff speed section of the animation is adjusted accordingly.

So far the goalkeeper can catch the ball in a plausible fashion in all target positions using the methods described.

6.7 Intelligence Level

One of the biggest advantages physics-based animation has over traditional animation is the interactive control over a character. For this project, where a virtual goalkeeper is created, adapting its intelligence level is relatively easy, due to the control we have over each mechanism. Currently, the goalkeeper is set to catch the ball perfectly in every target position. In real life the chances of this happening are low, sometimes the goalkeeper will catch the ball, sometimes they will block the ball and it will bounce away, and sometimes they will miss it completely. By adjusting the target position of each arm in the catching mechanism, these situations can be replicated. However, before adjusting the intelligence level, a metric is necessary to compare results.

6.7.1 Testing Framework

To measure a goalkeeper's intelligence, a testing framework can be put in place. If 100 shots are fired at the goalkeeper, and they save 90 of them, their save accuracy is 90%, meaning they are an intelligent goalkeeper.

To fire 100 shots and count the number of saves would be very time-consuming, therefore an automated system is created to fire 100 shots at various points in the goal. For each shot if the ball passes the goal line, it is considered a goal, if it does not pass the goal line, it is considered a save. This is checked by placing a box collider in the goal, and checking if the ball collides with it. A list of 100 target positions spread evenly throughout the goal is automatically generated and Unity's built-in test runner is used to fire the shots automatically.

6.7.2 Intelligence Level Adaption

To adapt the goalkeeper's intelligence level, the inverse kinematic target position for each arm can be offset by a small amount. This will decrease the chance the goalkeeper will catch and ball and increase the chance it slips through, or bounce off of the hands. To apply

this method, an offset distance can be applied to a random direction to move each inverse kinematic target. The offset distance will directly affect the goalkeeper's intelligence.

7 Results and Evaluation

7.1 Catching Mechanism

Once the catching system was implemented the goalkeeper was able to catch the ball in a plausible fashion. My implementation worked well, and the interpolation from the original animation to the inverse kinematic target position looked smooth and natural. Once the ball was caught, it was attached to one of the hands, this worked well because the goalkeeper could continue the rest of the animation while holding the ball in one hand.

One of the drawbacks of the catching mechanism was that the collision detection would sometimes not accurately calculate the contact point between the ball and the hand. This was a rare occurrence, and resulted in the ball being attached to the goalkeeper's hand in the incorrect place, giving the effect that the ball was hovering in the air, not the goalkeeper was holding onto it. Even with the goalkeeper's palms and the ball set to continuous collision detection, the problem still occurred. This could be due to inaccuracies in Unity's physics engine when detecting collisions.

Overall, I considered the catching system to be a simple, yet effective method to produce plausible results of a goalkeeper catching a ball.

7.2 Plausibility

7.2.1 Active Ragdoll

As mentioned in the Implementation section, plausible results were difficult to achieve using an active ragdoll setup. With the objective of creating a virtual, interactive, goalkeeper, the active ragdoll approach can achieve complete interactivity and is completely physics based with all limbs under the influence of physics.

My active ragdoll achieved plausible results for animations like idling, and walking, however, when applied to more complex animations like diving and jumping, the springy and floppy nature of the active ragdoll was highlighted resulting in animation which was not visually

natural. A plausible solution using an active ragdoll could have been possible, but due to the large number of hyperparameters and physical objects involved, reusable, repeatable results would of been unlikely.

While this method did not achieve the results I was looking for, a different approach, using principles from transition motion synthesis, which prioritises visually natural results was looked at.

7.2.2 Transition Motion Synthesis

Although the reconstruction of the animation curves method I used was not necessarily transition motion synthesis, the same principles and problems arise. How can an animation curve be reconstructed, connecting sections of the animation, while keeping the motion smooth and plausible?

The main aspects of the animation that were dynamically reconstructed using transition motion synthesis were:

1. Root Position
2. Launch Angle
3. Foot Placement
4. Animation Speed

This resulted in a relatively simple method of producing plausible animation for saving a ball in a target position based on a few reference animations. The reconstruction of each animation curve was based on the target position of the ball, and biometric characteristics of goalkeeping motion.

This method is not a physics simulation, it makes dynamics decisions based on physical principles while prioritising visually natural animation. The objective of this project was to create a virtual goalkeeper character that can save a ball fired at the goal in a plausible fashion, and this method achieves that goal, using a limited amount of motion capture data.

In most situations the goalkeeper will save the ball in a plausible fashion, however, in some cases, like when the target position is in the top corner, the take-off speed and trajectory does not match and look a little unnatural. Ideally, if more motion capture data was available, a reference animation of the goalkeeper taking an extra step before would allow for more natural-looking animations for saving balls at edge points of the goal.

7.3 Intelligence Levels

In video games, the intelligence levels of virtual characters is very important for player experience. If the artificially intelligent characters are too smart, the game becomes too difficult, and if the characters are not smart enough, the game is not challenging enough and becomes boring. If the goalkeeper in this project were to be implemented in a video game, ideally, the intelligence levels would be easily adjustable.

The following tests were run on the character with the transition motion synthesis techniques applied. The active ragdoll approach did not yield results good enough to test. In 7.1, the save accuracy was taken out of 100 shots. For each test run, the target position for the arms target position was offset by a constant amount in a random direction.

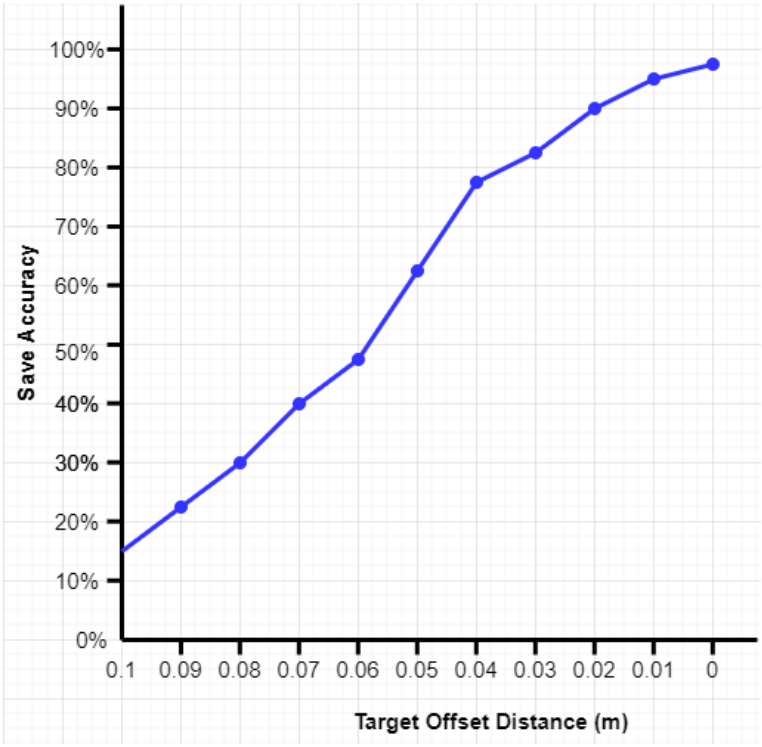

Figure 7.1: Target Offset Distance vs. Save Accuracy

The results from 7.1 display the feasibility of altering the virtual character's intelligence level while keeping plausible animation. Thanks to the physics-based methods used, the character's intelligence level can easily be adjusted to a desired amount.

8 Conclusion

8.1 Contribution

The animations of sporting motions have been heavily studied in the field of research. From walking, running, and jumping to acrobats, somersaults, and boxing. A virtual goalkeeper that can save a ball in a physically plausible manner is something that has not been studied so far, which is what intrigued me about this project. This project gave me the opportunity to learn and explore a variety of physics-based animation techniques, while also working on a virtual character that had not yet been studied. This forced me to come up with unique solutions to problems and analyse techniques used in different projects. Once I had researched potential methods, I implemented them, compared them, and analysed the results.

The combination of inverse kinematics, collision detection and pose-based animation to create a dynamic catching system resulted in a simple and natural-looking animation to catch and reach for the ball. This provided a solution for the goalkeeper to catch the ball when it was in range.

An active ragdoll approach was looked at to provide physically realistic results, however, did not produce plausible results. The torque applied to the rigid bodies attached to the character resulted in spring-like behaviour that did not look natural.

The dynamic reconstruction of animation curves based on the target location of the ball using the characteristics of goalkeeping motion is my proudest contribution to this project. This method takes advantage of the fact that only a short burst of dynamic motion is required to adapt the reference animation to reach its target position, and is perfect for the environment of a goalkeeper, a character that only interacts with the ball being fired at it. Due to limitations in Unity, an easy way to access and alter animation curves dynamically is not available. Therefore, my own system was designed to reconstruct the appropriate animation curves based on the target position of the ball and apply the animation. Since only the relevant animation curves are reconstructed, the time taken is efficient and the new animation is applied instantly once the ball is fired.

Many physics-based animation techniques were studied, evaluated, and implemented in this project. The result combines a variety of techniques, to construct a plausible physics-based character which can save the ball at any position in the goal. This system is also run in real-time, and is not computationally expensive.

8.2 Future Work

There are many aspects of this project which could be improved upon. Although plausible results were found, there are many different ways to approach this project.

1. **A more complex reconstruction system**

   The methods used to reconstruct the animation curves in this project were quite simple. A constant change was applied to a selection of key-frames to alter the animation curve. A more complex system could be utilised to create more dynamic motions, that strictly abide by the laws of physics.

2. **Active ragdoll Approach**

   Although I was not able to get plausible results using an active ragdoll, it is still a feasible approach, and once it is working is very interactive. Another aspect of the active ragdoll is to use imitation learning. Imitation learning utilises a reference animation and a reward system. A deep learning model is used to apply forces and is rewarded when it matches the reference animation. This is a hot topic of research today and could be utilised for a virtual goalkeeper.

Bibliography

[1] Wayne L. Wooten and Jessica K. Hodgins. Simulating leaping, tumbling, landing and balancing humans. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, 1:656–662 vol.1, 2000.

[2] J. Zhao, Y. Wei, S. Xia, and Z. Wang. Survey of physics-based character animation. 52:2866–2878, 12 2015. doi: 10.7544/issn1000-1239.2015.20140634.

[3] Thomas Geijtenbeek and Nicolas Pronost. Interactive character animation using simulated physics: A state-of-the-art review. *Computer Graphics Forum*, 31, 09 2012. doi: 10.1111/j.1467-8659.2012.03189.x.

[4] C. Liu and Dan Negrut. The role of physics-based simulators in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4, 05 2021. doi: 10.1146/annurev-control-072220-093055.

[5] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Transactions on Graphics*, 38: 1–13, 07 2019. doi: 10.1145/3306346.3322972.

[6] Yifeng Jiang, Tom Wouwe, Friedl De Groote, and Karen Liu. Synthesis of biologically realistic human motion using joint torque actuation. *ACM Transactions on Graphics*, 38:1–12, 07 2019. doi: 10.1145/3306346.3322966.

[7] Jesus Martinez-del Rincon, Jean-Christophe Nebel, Dimitrios Makris, and Carlos Orrite. Tracking human body parts using particle filters constrained by human biomechanics. 01 2008. doi: 10.5244/C.22.31.

[8] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics*, 36:1–13, 07 2017. doi: 10.1145/3072959.3073663.

[9] Sergey Levine and Jovan Popović. Physically plausible simulation for character animation. In *SCA '12*, 2012.

[10] Pawel Electronic, Arts Odest, Odest Jenkins, Morgan Mcguire, and Williams College. Dynamo: Dynamic, data-driven character control with adjustable balance. *Sandbox Symposium*, 08 2006. doi: 10.1145/1183316.1183325.

[11] Victor Zordan and Jessica Hodgins. Motion capture-driven simulations that hit and react. page 89, 01 2002. ISBN 1581135734. doi: 10.1145/545274.545276.

[12] Kwang Sok, Katsu Yamane, J. Lee, and Jessica Hodgins. Editing dynamic human motions via momentum and force. pages 11–19, 01 2010. doi: 10.2312/SCA/SCA10/011-019.

[13] Marco Silva, Jovan Popovic, and Yeuhi Abe. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics*, 27, 08 2008. doi: 10.1145/1399504.1360681.

[14] D. Sharon and M. Panne. Synthesis of controllers for stylized planar bipedal walking. volume 2005, pages 2387 – 2392, 05 2005. doi: 10.1109/ROBOT.2005.1570470.

[15] Victor Zordan, Anna Majkowska, Bill Chiu, and Matthew Fast. Dynamic response for motion capture animation. *ACM Trans. Graph.*, 24:697–701, 07 2005. doi: 10.1145/1073204.1073249.

[16] Seokpyo Hong, Daseong Han, Kyungmin Cho, Joseph Shin, and Junyong Noh. Physics-based full-body soccer motion control for dribbling and shooting. *ACM Transactions on Graphics*, 38:1–12, 07 2019. doi: 10.1145/3306346.3322963.

[17] Keita Matsukura, Takeshi Asai, and Keiko Sakamoto. Characteristics of movement and force exerted by soccer goalkeepers during diving motion. *Procedia Engineering*, 72: 44–49, 12 2014. doi: 10.1016/j.proeng.2014.06.010.

[18] Herbert Reynolds, Charles Clauser, John Mcconville, Richard Chandler, and J. Young. Mass distribution properties of the male cadaver. *SAE (Society of Automotive Engineers) Transactions*, SAE 750424, pp. 1132-1150, 02 1975. doi: 10.4271/750424.

[19] Stevenson Michael. Configurable joint extensions. `https://gist.github.com/mstevenson/7b85893e8caf5ca034e6`, 2014.